SELECTED EXAMPLES OF COMPUTER AIDED SOLUTIONS FOR MANAGEMENT PROBLEMS

A THESIS

SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

BY

THAWEESAK CHAMCHOI

DEPARTMENT OF MATHEMATICS

ATLANTA, GEORGIA

MAY 1982

ABSTRACT

COMPUTER SCIENCE

CHAMCHOI, THAWEESAK B.S., KASETSAKT UNIVERSITY, 1978

Selected Examples of Computer Aided Solutions for Management Problems

Advisor: Dr. Nazir A. Warsi

Thesis dated April 23, 1982

This thesis explains three algorithms useful in management science for decision making. These are: critical path method, northwest corner rule and decision tree evaluation. Illustrative examples and complete PASCAL programs are included for each.

ACKNOWLEDGEMENTS

I wish to thank Dr. Nazir A. Warsi and Dr. Ben Setzer for their valuable suggestions in the completion of this thesis. I also wish to thank all those who read drafts of the manuscript, pointed out mistakes to me, and made constructive suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

The duties of a modern manager involve decision making for a variety of day-to-day problems. Computers can be of great aid in the managerial process because computers can systematically and routinely analyze complex problems with a large number of variables.

In this project, there are three selected examples described: The Critical Path Method, The Northwest Corner Rules and The Decision Tree. In order to facilitate readability, each of these method is described separately in a self-contained manner.

Chapter I : Deals with
The Critical Path Method.

Chapter II : Deals with
The Northwest Corner Rules.

Chapter III : Deals with
The Decision Tree.

For each one of the methods discussed in the subsequent chapters, a complete computer algorithm is described and programmed on a PDP 11/40, in PASCAL. Also, algorithm analyses including space and time complexities are discussed.

For space complexity, only the memory reserved by the array declarations is considered. Since variables add

only a constant number to this complexity, they are ignored in this analysis.

For time complexity, the assumption is made that all arithmetic operations and assignments take unit times. Note that the polynomial time complexity of order k of an algorithm with input parameter N is $O(N^k)$.

It is hoped that these examples adequately illustrate how the power and speed of modern computers can be used to solve management problems.

CHAPTER I

THE CRITICAL PATH METHOD

1.1 Basic Concepts

The critical path method (CPM) describes a network which is used to determine the earliest and the latest times when activities within a project may start if the project is to finish in the shortest possible time. CPM is of an interest to the system analyst for four reasons.

1. It may provide the solution to a client's problem.

2. It may be an important element in a tailor-made system designed to solve a client's problem.

3. It forms the basis for more elaborate algorithms, such as resource-constrained scheduling methods.

4. It may be used by the systems analysts to manage his own activities.[1]

1.2 Project Networks

CPM can be used whenever a project can be partitioned into activities, each with a fixed duration. Included in this context would be some specifications regarding the

---

[1] Arne Thsen, Computer Methods in Operations Research (New York: Academic Press, Inc., 1978), p. 103.

completion of requisite (predecessor) activities prior to the start of each activity.

The project can be represented as a network where nodes represent activities and links represent precedence requirements. An alternative representation associating activities with areas and nodes with points in time is also used.

To illustrate the concept of a CPM diagram, consider the representation in Table 1. Fig. 1 shows how the resulting CPM network looks in activities-on-nodes notation. The corresponding network in activities-on-links notation is shown in Fig. 2.

Algorithms to calculate the earliest start times for individual activities are based on the premise that an activity cannot start until all its predecessor activities have been completed. In fact, the earliest start time for an activity is the exact moment when the last of the activity's predecessors finished. In terms of a network consider all chains from the start node to the current acitity node where these chains are defined by the duration of the activities. Then the earliest start time for an activity is defined by the longest chain from the start node to that activity node. After calculating all the earliest start times, the project duration is fixed as the earliest time for the finished event. The latest start time for all activities is then calculated in a similar manner passing

TABLE 1

DIAGRAM OF PROJECT NETWORK

LABEL	ACTIVITY	DURATION	IMMEDIATE PREDECESSOR
A	Start	0	None
B	Dig Hole	2	A
C	Pour Foundation	3	D,E
D	Order Concrete	1	A
E	Construct Forms	1	B,H
F	Remove Forms	4	C
G	Step	0	F
H	Order Lumber	2	A

FIGURE 1

ACTIVITIES-ON-NODES NOTATION

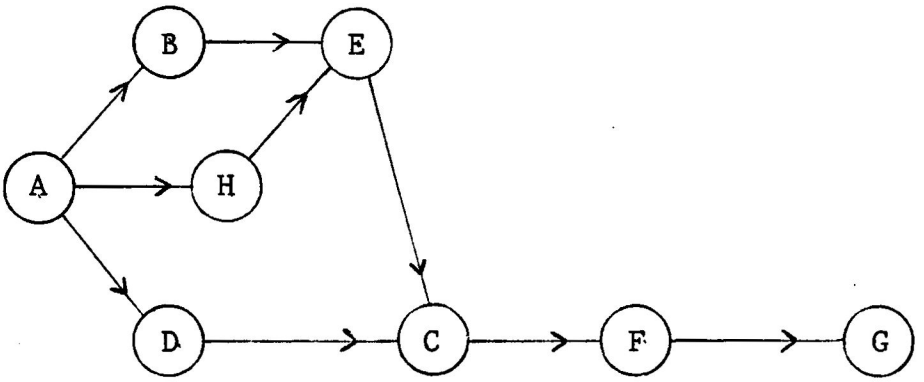
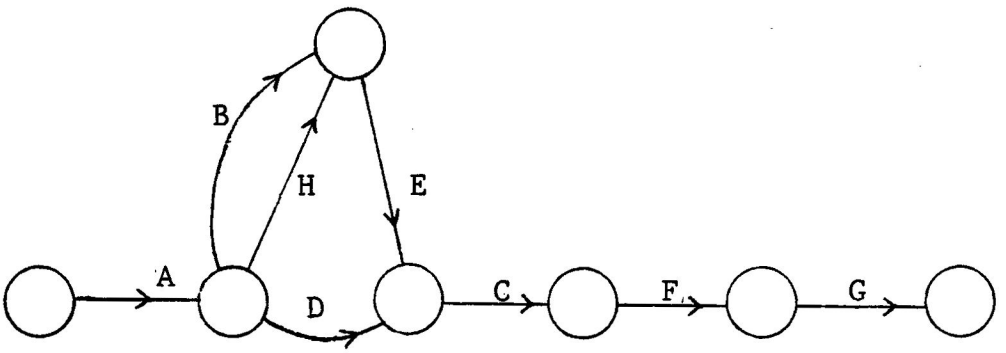


FIGURE 2

ACTIVITIES-ON-LINKS NOTATION



backward over the network.

1.3 Critical Path Method Algorithm

Step 1: Develop a list of activities that make up the project including immediate predecessors.

Step 2: Draw a network corresponding to the activity list developed in step 1.

Step 3: Estimate the completion time for each activity.

Step 4: Using the completion times, determine the earliest finish time (EF) for each activity. The earliest finish time (EF) for the complete project corresponds to the last activity. This is the expected project completion time.

Step 5: Using the project completion time at the last activity, work backward through the network to compute the last start time (LS) and last finish time (LF) for each activity.

Step 6: Compute the slack associated with each activity (slack = LS-ES=LF-EF for each activity). The critical path activities are the activities with zero slacks.[2]

---

[2]David R. Anderson, Dennis J. Sweeney, Thomas A. Williams, An Introduction to Management Science (St. Paul, Minnesota: West Publishing Co., 1979), pp. 381-383.

1.4 Computer Algorithm for Critical Path Method

By using the logical idea from the critical path method algorithm, the programming can be broken into five steps: (1) initialize path table, (2) generate the paths of activities and completion times for each activity by data input, (3) calculate the earliest start and the earliest finish times for each activity, (4) calculate the latest start and the latest finish times for each activity, (5) calculate slacks then print out the result. Each step is described as a procedure. Only the calculation steps will be shown by the list of procedures.

1.4.1 Specified Variables

The specified variables used are listed below:

Variables		Descriptions
N	=	Number of activity.
SLACK	=	Slack time.
NODE [I]	=	Array of activity (task) names.
TASKS [I, n]	=	Associated times for the I^{th} activity where n = 0 to 4.
TASKS [I, 0]	=	Time needed for completion of a task.
TASKS [I, 1]	=	The earliest start time.
TASKS [I, 2]	=	The earliest finish time.
TASKS [I, 3]	=	The latest start time.
TASKS [I, 4]	=	The latest finish time

1.4.2 Space Complexity

Theorem : The space complexity of the critical path method is $O(N^2)$ where N is the number of tasks.

Proof : The space complexity can be computed from Table 2.

TABLE 2

VARIABLES TABLE CPM

[0..N] OF [1..16] OF CHAR : NODES	16(N+1)
[0..N] OF [0.. N] OF BOOLEAN : PATH	$(N+1)^2$
[0..N, 0..4] OF REAL : TASKS	5(N+1)
	$N^2 + 23N + 3$

The total shows that the space complexity of the critical path algorithm is $O(N^2)$ where N is number of activities.

1.4.3 Time Complexity

Theorem : The time complexity of CPM algorithm (ignoring input and output parts) is given by $O(N^2)$ where N is the number of activities

Proof : The steps of algorithm for calculation of time complexity are shown by step3 and step 4. The list of calculations and the time complexity

are listed in Table 3.

Step 3 : Starting from the first activity to the last activity, calculate the earliest start and the earliest finish time.

```
PROCEDURE CALCULATE START TIME;
  BEGIN
   FOR  I :=1 TO N DO
  ┌─ BEGIN
  │    EARLY  ←  0.0;
  │  ┌─FOR j:=1 TO N DO
1₂│1₁│   IF  PATH  [I, j]  THEN
  │  │   IF  EARLY < TASKS  [j, 2]  THEN
  │  └       EARLY ← TASKS  [j, 2]  ;
  │         TASKS  [I, 1]  ←  EARLY;   /* ES */
  │         TASKS  [I, 2]  ←  EARLY + TASK [I,0];/*EF*/
  └─END;
   END;  /* END OF PROCEDURE */
```

Step 4: Starting from the last activity to the first activity, calculate the latest start and the latest finish time.

```
PROCEDURE  LATETIME;
  BEGIN
   STIME  ←—  TASKS  [N, 1]  ;
   FOR  I : = N DOWNTO 1 DO
   ┌BEGIN
   │  EARLY ← STIME;
1  │  ┌FOR j : = 1 TO N DO
 2 │  │  IF PATH  [j, I]  THEN
   │  │  IF EARLY > TASKS  [j, 3]  THEN
   │1 └     EARLY ← TASKS  [j, 3]  ;
   │1        TASKS  [I, 4] ←— EARLY ; /*LF*/
   │         TASKS  [I, 3] ←— EARLY - TASKS [I,0] ;/*LS*/
   └END;
   END;  /*  END OF PROCEDURE */
```

TABLE 3

CPM TIME COMPLEXITY

PROCEDURE	LEVEL	TYPE OF LOOP	RANGE	ASSIGNMENTS + OPERATION	RESULT	COMMENTS
START TIME	1_1	FOR	1..N	1	N	N = number of
START TIME	1_2	FOR	1..N	$5 + 1_1$	$6N^2$	activities
LATE TIME	1_1	FOR	1..N	1	N	
LATE TIME	1_2	FOR	N..1	$5 + 1_1$	$6N^2$	
* TOTAL RESULT					$12N^2$	

The total result shows that the time complexity of the Critical path algorithm is $O(N^2)$ where N is the number of activities.

```
/*
                Program Documentation
                ---------------------

        Program :       CPM.PAS
        Author  :       Thaweesak Chamchoi
        Purpose :       Critical Path Method
      Critical Path Method : Calculates the critical path,that is,the
    sequence of events whose completion times are most critical to
    the finishing time of a set of events.

                Variable List
                -------------

ALPHA           -       Variable type used to denote string array of size 16
EARLY           -       Earliest calculated time for a task
FOUND           -       Test Flag for task name found
I               -       Index variable
J               -       Index variable
LATE            -       Latest calculated starting time for a task
NODES           -       Array of task names
NOTASKS         -       Number of tasks
PATHS           -       Matrix containing which tasks are connected
PREV            -       No. of predecessors to a task
PREVTASK        -       Name of previous task
SLACK           -       Amount of spare time for a task
SPACE           -       Denotes a set of storage(i.e. 0..50)
STIME           -       Start time for a task
TASKS           -       Times associated with any task
                        TASKS[n,0] - Time need to complete a task
                        TASKS[n,1] - Earliest start time
                        TASKS[n,2] - Earliest finish time
                        TASKS[n,3] - Latest start time
                        TASKS[n,4] - Latest finish time

                Procedure List
                --------------

        Functions : None
        Procedures
                CALCULATESTARTTIMES -  Calculate starting times
                INITIALIZE          -  Initializes path matrix to false
                LATETIMES           -  Calculates Late times
                PATH                -  Main procedure call to all others
                PRINTOUT            -  Print data in table form
                READDATA            -  Read data and connecting tasks
 Critical Path Method : Calculates the critical path,that is,the
   sequence of events whose completion times are most critical to
   the finishing time of a set of events.

Data is inputted as follows :
1st line of input  - number of tasks in problem

Then for each task is input
(1) A task name
(2) No. of predecessors
    If the number of predecessors is not zero, then
    for each predecessor we must input
      (3) Task name of predecessor (already have been input)
*/
```

```
  PROGRAM CRITICALPATHMETHOD(INPUT,OUTPUT);
TYPE
  SPACE=0..50;
  ALPHA=ARRAY[1..16] OF CHAR;
VAR
  EARLY,LATE,STIME,SLACK:REAL;
  PREV,I,I1,J,NOTASKS:INTEGER;
  FOUND:BOOLEAN;
  PATHS:ARRAY[SPACE,SPACE] OF BOOLEAN;
  NODES:ARRAY[SPACE] OF ALPHA;
  TASKS:ARRAY[SPACE,0..4] OF REAL;
  PREVTASK:ALPHA;
  PROCEDURE PATH;
  LABEL
    999;
    PROCEDURE READDATA;
     BEGIN
       WRITE(' NO. OF TASKS ');
       READLN(NOTASKS);
       /*
        GET DATA
        */
       FOR I:=1 TO NOTASKS DO
        BEGIN
          WRITELN('TASK #':20,I:5);
          WRITE(' TASK NAME ');
          READLN(NODES[I]);
          WRITE(' PRECEDING TASK # ');
          READLN(PREV);
          /*  Search for previous name to set up links of events sequence */
          FOR J:=1 TO PREV DO
           BEGIN
             WRITE(' PREV. TASK NAME ');
             READLN(PREVTASK);
             I1:=0;
             FOUND:=FALSE;
              REPEAT
               I1:=I1+1;
               FOUND:=PREVTASK=NODES[I1];
              UNTIL (FOUND) OR (I1=I-1);
             IF NOT FOUND
              THEN GOTO 999;
             PATHS[I,I1]:=TRUE;
           END;
          WRITE(' NO. OF DAYS FOR COMPLETION ');
          READLN(TASKS[I,0]);
        END;
     END;
     PROCEDURE INITIALIZE;
      /*  Initialize all links to be null */
      BEGIN
        FOR I:=1 TO 50 DO
        FOR J:=1 TO 50 DO
        PATHS[I,J]:=FALSE;
      END;
```

```
PROCEDURE CALCULATESTARTTIMES;
 BEGIN
   FOR I:=1 TO NOTASKS DO
    BEGIN
      EARLY:=0.0;
      /*   Use the latest start times for the event */
      FOR J:=1 TO NOTASKS DO
      IF PATHS[I,J]
       THEN
       IF EARLY < TASKS[J,2]
        THEN EARLY:=TASKS[J,2];
      TASKS[I,1]:=EARLY;
      TASKS[I,2]:=EARLY + TASKS[I,0];
    END;
 END;
/* END OF PROCEDURE */
PROCEDURE LATETIMES;
 BEGIN
   STIME:=TASKS[NOTASKS,2];
   FOR I:=NOTASKS DOWNTO 1 DO
    BEGIN
      EARLY:=STIME;
      FOR J:=1 TO NOTASKS DO
      IF PATHS[J,I]
       THEN
       IF EARLY > TASKS[J,3]
        THEN EARLY:=TASKS[J,3];
      TASKS[I,4] := EARLY;
      TASKS[I,3] := EARLY - TASKS[I,0];
    END;
   /* FOR */
 END;
/* PROCEDURE */
PROCEDURE PRINTOUT;
 BEGIN
   WRITELN('TIME OF COMPLETION = ',STIME:9:2);
   WRITELN;
   WRITE('Task':5,'TIME':12,'Early':12);
   WRITE('Latest':12,'Early':13,'Latest':13);
   WRITELN('Slack':8);
   WRITE('NEEDED':17,'START':12,'START':12);
   WRITELN('FINISH':12,'FINISH':12);
   WRITELN('ES':27,'LS':12,'EF':12,'LF':12);
   WRITELN;
   FOR I:= 1 TO NOTASKS DO
    BEGIN
      SLACK:=TASKS[I,3]-TASKS[I,1];
      WRITE(NODES[I]:8,TASKS[I,0]:10:2,TASKS[I,1]:12:2);
      WRITE(TASKS[I,2]:12:2,TASKS[I,3]:12:2,TASKS[I,4]:12:2);
      WRITELN(SLACK:10:2);
    END;
 END;
```

```
(* MAIN PROGRAM *)
   BEGIN
     INITIALIZE;
     READDATA;
     CALCULATESTARTTIMES;
     LATETIMES;
     PRINTOUT;
999:
     IF NOT FOUND
      THEN
      WRITELN('DATA OUT OF ORDER - REDO');
   END;
 BEGIN
   PATH;
 END.
```

```
NO. OF TASKS  11
              TASK #     1
TASK NAME START
PRECEDING TASK # 0
NO. OF DAYS FOR COMPLETION 0
              TASK #     2
TASK NAME BLUEPRINT
PRECEDING TASK # 1
PREV. TASK NAME START
NO. OF DAYS FOR COMPLETION 5
              TASK #     3
TASK NAME LABOR
PRECEDING TASK # 1
PREV. TASK NAME START
NO. OF DAYS FOR COMPLETION 7
              TASK #     4
TASK NAME ORDER
PRECEDING TASK # 1
PREV. TASK NAME START
NO. OF DAYS FOR COMPLETION 14
              TASK #     5
TASK NAME CHANGE
PRECEDING TASK # 2
PREV. TASK NAME BLUEPRINT
PREV. TASK NAME LABOR
NO. OF DAYS FOR COMPLETION 14
              TASK #     6
TASK NAME FABRICATION
PRECEDING TASK # 2
PREV. TASK NAME BLUEPRINT
PREV. TASK NAME LABOR
NO. OF DAYS FOR COMPLETION 10
              TASK #     7
TASK NAME MIX
PRECEDING TASK # 1
PREV. TASK NAME ORDER
NO. OF DAYS FOR COMPLETION 6
              TASK #     8
TASK NAME TEST
PRECEDING TASK # 1
PREV. TASK NAME FABRICATION
NO. OF DAYS FOR COMPLETION 5
              TASK #     9
TASK NAME PRINT
PRECEDING TASK # 2
PREV. TASK NAME TEST
PREV. TASK NAME MIX
NO. OF DAYS FOR COMPLETION 1
              TASK #    10
TASK NAME COMPLETE
PRECEDING TASK # 2
PREV. TASK NAME CHANGE
PREV. TASK NAME PRINT
NO. OF DAYS FOR COMPLETION 7
              TASK #    11
TASK NAME END
PRECEDING TASK # 1
PREV. TASK NAME COMPLETE
NO. OF DAYS FOR COMPLETION 0
```

TIME OF COMPLETION = 30.00

Task	TiME NEEDED	Early START ES	Latest START LS	Early FINISH EF	Latest FINISH LF	Slack
START	0.00	0.00	0.00	0.00	0.00	0.00
BLUEPRIN	5.00	0.00	5.00	2.00	7.00	2.00
LABOR	7.00	0.00	7.00	0.00	7.00	0.00
ORDER	14.00	0.00	14.00	2.00	16.00	2.00
CHANGE	14.00	7.00	21.00	9.00	23.00	2.00
FABRICAT	10.00	7.00	17.00	7.00	17.00	0.00
MIX	6.00	14.00	20.00	16.00	22.00	2.00
TEST	5.00	17.00	22.00	17.00	22.00	0.00
PRINT	1.00	22.00	23.00	22.00	23.00	0.00
COMPLETE	7.00	23.00	30.00	23.00	30.00	0.00
END	0.00	30.00	30.00	30.00	30.00	0.00

>

CHAPTER II

THE NORTHWEST CORNER RULES

2.1 Basic Concepts

Transportation problems are concerned with the transportation of goods from suppliers to destinations. With a variety of shipping routes and a variety of costs for the routes, one must determine how many units should be shipped from each origin to each destination, so that the demands of all destinations are satisfied and the total transportation costs are minimized.

We use the following notations to describe a transportation problem:

X_{ij} = amount shipped from i to j

i = 1 , 2 , ... m ;

j = 1 , 2 , ... n .

C_{ij} = cost of shipping one unit from supplier i to destination j.

i = 1 , 2 , ... m ;

j = 1 , 2 , ... n .

D_j = required number of unit at destination j.

j = 1 , 2 , ... n .

Si = capacity of supplier i.

i = 1 , 2 , ... m .

This problem reduces to minimizing.

$$Z \min = \sum_{i=1}^{m} , \sum_{j=1}^{n} C_{ij} * X_{ij}$$

subject to

$$\sum_{j=1}^{m} X_{ij} \leqslant S_i \quad : i = 1 , 2 , ... m.$$

$$\sum_{i=1}^{n} X_{ij} \geqslant D_j \quad : j = 1 , 2 , ... n.$$

$$\text{All } X_{ij} \geqslant 0.$$ [3]

2.2 The Northwest Corner Rules

The northwest corner rules is a method for identifying an initial feasible solution: start the solution in the upper left-most corner of X_{ij} matrix and allocate units to routes in a "move-to-right-and-down" fashion.[4] The method attempts to reduce the amount of supply units (S_i) and set zero values for them. The feasible solution is found when all demands (D_j) and suppliers (S_i) have values zero.

The major advantage of using the northwest corner rules for finding an initial feasible solution is that it is

---

[3] Billy E. Gillett, Introduction to Operations Research (New York: McGraw-Hill Book Company, 1976), pp. 107-108.

[4] David R. Anderson, Dennis J. Sweeney, Thomas A. Williams, An Introduction to Management Science (St. Paul, Minnesota: West Publishing Co., 1979), pp. 294-295.

fast and easy to use. However, the initial feasible solution obtained using this rule will not be very good since the cost of shipping over each of the routes is totally ignored. Vogel's approximation method, the stepping-stone method, modified distribution method and simplex method are others method used to solve the transportation problem with minimial cost. The output from the northwest corner rules is input for other complex algorithm.

2.3 The Northwest Corner Rules Algorithm

Step 1: Assign $i = j = 1$

Step 2: Using the i,j cell of the X_{ij} matrix assign as many units to this cell as possible (minimum of S_i and D_j).

Step 3: Reduce the row supply and column demand by the amount assigned to the cell.

Step 4: If the row supply is now zero then ($i \leftarrow i+1$), move down the column to the next cell; if the column demand is now zero then ($j \leftarrow j+1$), move to the right along the row to the next cell; if both the row supply and column demand are zeros, move down one cell and to the right one cell to the next cell.

Step 5: If the total supply has not been exhausted: GO TO STEP 3.

2.4 Computer Algorithm for Northwest Corner Rules

Based on the basic concept and the algorithm, the computer programming can be developed in 3 steps as follow: (1) create table of demand units, supply units and transportation costs associated between cities. (2) use of the northwest corner rules to calculate the distribution. (3) calculate the total cost by summation of the transportation cost times * the distribution. Each step is then implemented as a procedure. Only the calculation steps will be shown for computation of time complexity. The complete work and execution results are shown in the program listing.

The variables used in this programming are represented by a 3 dimension array to aid in the print out of the table of work.

2.4.1 Specified Variables

The specified variables used are shown in table 4.

2.4.2 Space Complexity

Theorem : The space complexity of the northwest corner rules is given by $O(N^2)$ where N is the maximum number of supply or demand centers.

Proof : The space complexity can be calculated from table 5.

TABLE 4

VARIABLE LIST FOR THE NORTHWEST CORNER RULES

VARIABLE	MODEL REFERENCE	DESCRIPTIONS
NDEST	m	Number of demand centers.
NSUPPLY	n	Number of supply centers.
TOTAL	Zmin	Total minimize cost.
TOTNEED	$\sum_{i=1}^{n} X_{ij}$	Total demand units.
TOTSUPPLY	$\sum_{j=1}^{m} X_{ij}$	Total supply units.
TABLEAU [I , 0 , 2]	S_i	Capacity of supply.
TABLEAU [I , n , 2]	D_j	Capacity of demand.
TABLEAU [I , j , 2]	X_{ij}	The distribution of the supply from the I^{th} suppliers to j^{th} centers.
TABLEAU [I , j , 1]	C_{ij}	The travel cost per unit from I^{th} suppliers to j^{th} destination.

TABLE 5

VARIABLES TABLE FOR THE NORTHWEST CORNER RULES

VARIABLE		TOTAL
[1..N,0..N , 1..2]	OF INTEGER: TABLEAU	2N(N+1)
[1..N] OF [1..20]	OF CHAR : SUPPLIERS, CENTER	20N
		$2N^2+22N$

The total shows that the space complexity of the northwest corner rules is $O(N^2)$ where N is the maximum number of supply or demand centers.

2.4.3 Time Complexity

Theorem : The time complexity of the northwest corner rules (ignoring input and output parts) is given by O(ND+NS) where ND is the number of demand centers and Ns is the number of supply centers.

Proof : The steps of algorithm for calculation of time complexity are shown by step 2.
The list of calculation and result of time complexity are listed in table 6.

Step 2: Using the northwest corner rules reduces the total

cost unit to zero.

```
     PROCEDURE SOLUTION ;
      BEGIN
         I ← 1 ;
         j ← 1 ;
        WHILE TOTSUPPLY > 0 THEN
         BEGIN
         PRINTTABLEAU ;
        IF TABLEAU [I,0,2] < TABLEAU [j,N1,2] THEN
         BEGIN
          K ← TABLEAU [I,0,2]  ;
    1₁    TABLEAU [I,j,2] ← K ;
          TABLEAU [j,N1,2] ← TABLEAU [j,N1,2]  -K ;
          TABLEAU [I,0,2] ← 0 ;
          I ← I+1 ;
  1₃     END ;
           ELSE
           BEGIN
            K ← TABLEAU [j,N1,2]  ;
           TABLEAU [I,j,2] ← K ;
    1₂     TABLEAU [I,0,2] ← TABLEAU [I,0,2] -K ;
           TABLEAU [j,N1,2] ← 0 ;
           j ← j+1 ;
         END ;  /* IF */
            TOTSUPPLY ← TOTSUPPLY - K ;
            TOTNEED  ←  TOTNEED - K ;
        END ;  /* WHILE */
      END ;  /* END OF PROCEDURE */
```

TABLE 6

TIME COMPLEXITY FOR THE NORTHWEST CORNER RULES

PROCEDURE	LEVEL	TYPE OF LOOP	RANGE	ASSIGNMENT & OPERATION	RESULT	COMMENT
SOLUTION	1_1	IF	—	7	7	It takes (ND*NS) time to initialize TABLEAU [I,j,2] to ∅.
SOLUTION	1_2	ELSE	—	7	7	The actual time spent by solution is ND+NS as i teration at 1_3 makes at least one move 'right' or one move 'down.'
SOLUTION	1_3	WHILE	—	$\frac{1_1 + 1_2 + 4}{2}$	11(ND+NS)	
*TOTAL RESULT					11(ND+NS)	

The total shows that the time complexity of the northwest corner rules is O(ND+NS) where ND is the number of demand centers and NS is the number of supply centers.

```
/*
P r o g r a m  D o c u m e n t a t i o n
---------------------------------------

Author  :       Thaweesak Chamchoi
Program :       TRANSP.PAS

V a r i a b l e  L i s t
-----------------------

CENTER          -       Array of Suppliers' Names
COST            -       Matrix of transportation costs
I               -       Index Variable
J               -       Index Variable
K               -       Index Variable
N1              -       Work Variable
NAME            -       Type, denotes a string of size 16
NDEST           -       Number of destinations
NSUPPLY         -       Number of suppliers
SUPPLIERS       -       Array of Suppliers' Names
TABLEAU         -       Matrix of transportation distribution
TOTAL           -       Total cost for transportation
TOTNEED         -       Total supply
TOTSUPPLY       -       Total supply from suppliers

P r o c e d u r e s
-------------------

READDATA        -       Procedure to read total need
and supply checking for equal
amounts.

SOLUTION        -       Northwest Corner Rule

TOTALCOST       -       Prints the cost of the TOTAL Distribution
Program : Transportation problem
That is, given M suppliers and N destinations, what
distribution will allow a low cost, and feasible solution
given the costs of transportation.

Method Used : Northwest Corner Rule

Comments :      Although this method will find a solution extremely fast
the solution can be bettered or worsened by a juxtaposition
of the input.

                I N P U T  P A R A M E T E R S
                ------------------------------

    1ST NUMBER IS NUMBER OF SUPPLIERS
      FOR EACH SUPPLIER WE THEN INPUT A NAME (<=20 CHARS)
      AND A PRODUCTION CAPACITY.

   NEXT SET OF DATA IS FIRST THE NUMBER OF DISTRIBUTORS
      FOR EACH DISTRIBUTORS WE THEN INPUT A NAME (<=20 CHARS)
      AND A DEMAND CAPACITY.

       N O T E !!! : DATA IS CHECK FOR EQUAL CAPACITY FOR SUPPLY
                     AND DEMAND.



   NEXT SET OF DATA IS TRANSPORTATION COST FROM EACH SUPPLIER
   TO EACH DISTRIBUTOR.
*/
```

```
  PROGRAM TRANSPORTATION (INPUT,OUTPUT);

TYPE
  NAME     = ARRAY[1..20] OF CHAR;
VAR
  N1,I,J,K,NSUPPLY,NDEST,TOTSUPPLY,TOTNEED: INTEGER;
  COST,TOTAL:REAL;
  TABLEAU : ARRAY[1..10,0..11,1..2] OF INTEGER;
  SUPPLIERS,CENTER : ARRAY[1..10] OF NAME;

  /*       Get the data and check original totals for proper sums */
  PROCEDURE READDATA;
   BEGIN
      REPEAT
       WRITELN('INPUT SUPPLIERS AND THEIR CAPABILITES');
       WRITELN;
       WRITE('NO. OF SUPPLIERS');
       TOTSUPPLY := 0;
       READLN(NSUPPLY);
       FOR I:= 1 TO NSUPPLY DO
        BEGIN
          WRITELN(' SUPPLIERS #',I:3);
          WRITE('NAME ');
          READLN(SUPPLIERS[I]);
          WRITE(' PRODUCTION CAPACITY ');
          READLN(TABLEAU[I,0,2]);
          TOTSUPPLY := TOTSUPPLY+TABLEAU[I,0,2];
        END;

       WRITELN(' INPUT DESTINATION PLANT AND THEIR DEMAND');
       WRITELN;
       WRITE(' NO. OF DISTRIBUTION CENTERS ');
       READLN(NDEST);
       TOTNEED := 0;
       FOR I:= 1 TO NDEST DO
        BEGIN
          WRITELN('CENTER #',I:3);
          WRITE('NAME ');
          READLN(CENTER[I]);
          WRITE('DEMAND ');
          READLN(TABLEAU[I,NDEST+1,2]);
          TOTNEED := TOTNEED+TABLEAU[I,NDEST+1,2];
        END;

       IF TOTNEED<>TOTSUPPLY
        THEN
        WRITELN('DEMAND IS NOT EQUAL SUPPLY PLEASE REDO');
     UNTIL  TOTNEED = TOTSUPPLY;

    WRITELN('INPUT TRANSPORTATION COST');
    WRITELN;
    FOR I:= 1 TO NSUPPLY DO
     BEGIN
       WRITELN('FROM ',SUPPLIERS[I]);
       FOR J:=1 TO NDEST DO
        BEGIN
          WRITE('TO ',CENTER[J]);
          READLN(TABLEAU[I,J,1]);
        END;
     END;



  END;
  /* READDATA */
```

```
/*      Doing execution print the reduced cost matrix as a means of
 showing the decision steps.
 */

PROCEDURE PRINTTABLEAU;
VAR
  I,J,K:INTEGER;
 BEGIN
   FOR I:=1 TO (NSUPPLY+2)*10 DO WRITE('.'); WRITELN;
   WRITE('From');
   I:=(10*NSUPPLY+26) DIV 2;
   WRITE('To Destination':I);
   I:=10*(NSUPPLY+2)-I;
   WRITE(' ':I+2,'Supply');
   WRITELN;
   WRITE(' ':10);
   FOR I:= 1 TO NDEST DO
   WRITE(CENTER[I]:10);
   WRITELN;
   FOR I:= 1 TO NSUPPLY DO
    BEGIN
      WRITE(SUPPLIERS[I]:10);
      FOR J:= 1 TO NDEST DO
      WRITE(TABLEAU[I,J,1]:10);
      WRITELN;
      WRITE(' ':10);
      FOR K:= 1 TO NDEST DO
      WRITE(TABLEAU[I,K,2]:10);
      WRITELN(TABLEAU[I,0,2]:10);
      WRITELN;
    END;
   WRITE(' ':10);
   FOR J:= 1 TO NDEST DO
   WRITE(TABLEAU[J,N1,2]:10);
   WRITELN(TOTNEED:10);
   FOR I:=1 TO (NSUPPLY+2)*10 DO WRITE('.'); WRITELN;
   WRITELN;
 END;
```

```
/*      SOLUTION : Northwest Corner Rule
 Reduction of the cost matrix and associated
 supplies by determining minimum between a
 supplier and a destination and going to a
 new supplier or destination depending on which
 has been reduced to zero.
 */
PROCEDURE SOLUTION;
 BEGIN
   I:=1;
   J:=1;
     N1:=NODEST+1;
   WHILE TOTSUPPLY>0 DO
    BEGIN
      PRINTTABLEAU;
      IF TABLEAU[I,0,2] < TABLEAU[J,N1,2]
       THEN
        BEGIN
          K:=TABLEAU[I,0,2];
          TABLEAU[I,J,2]:=K;
          TABLEAU[J,N1,2]:=TABLEAU[J,N1,2]-K;
          TABLEAU[I,0,2]:=0;
          I:=I+1;
        END
       ELSE
        BEGIN
          K:=TABLEAU[J,N1,2];
          TABLEAU[I,J,2]:=K;
          TABLEAU[I,0,2]:=TABLEAU[I,0,2]-K;
          TABLEAU[J,N1,2]:=0;
          J:=J+1;
        END;
      TOTSUPPLY:=TOTSUPPLY-K;
      TOTNEED:=TOTNEED-K;
    END;
   /*WHILE */
 END
  ;
/* SOLUTION */
```

```
 /* Print out solution with calculated costs and total.  */
 PROCEDURE TOTALCOST;
  BEGIN
    WRITELN('ROUTE':22,'UNIT':22,'PER UNIT':11,'TOTAL':11);
    WRITELN('FROM ','TO':19,'SHIPPED':20,'COST':9,'COST':13);
    TOTAL := 0.0;
    FOR I:=1 TO NSUPPLY DO
    FOR J:= 1 TO NDEST DO
    IF TABLEAU[I,J,2]>0
     THEN
      BEGIN
        COST := TABLEAU[I,J,1]*TABLEAU[I,J,2];
        TOTAL:=TOTAL+COST;
        WRITE(SUPPLIERS[I]:19,CENTER[J]:19);
        WRITELN(TABLEAU[I,J,2]:6,TABLEAU[I,J,1]:8,COST:19:3);
      END;
    WRITELN('TOTAL':56,TOTAL:10:0);
  END;
 /*TOTALCOST */

BEGIN
  READDATA;
  SOLUTION;
  TOTALCOST;
END.
/* END OF PROGRAM TRANSPORTATION */
```

```
INPUT SUPPLIERS AND THEIR CAPABILITES

NO. OF SUPPLIERS 3
 SUPPLIERS #  1
NAME CLEVELAND
 PRODUCTION CAPACITY 5000
 SUPPLIERS #  2
NAME BEDFORD
 PRODUCTION CAPACITY 6000
 SUPPLIERS #  3
NAME YORK
 PRODUCTION CAPACITY 2500
 INPUT DESTINATION PLANT AND THEIR DEMAND

 NO. OF DISTRIBUTION CENTERS 4
CENTER #  1
NAME BOSTON
DEMAND 6000
CENTER #  2
NAME CHICAGO
DEMAND 4000
CENTER #  3
NAME ST.LOUIS
DEMAND 2000
CENTER #  4
NAME LEXINGTON
DEMAND 1500


INPUT TRANSPORTATION COST

FROM CLEVELAND
TO BOSTON              3
TO CHICAGO             2
TO ST.LOUIS            7
TO LEXINGTON           6
FROM BEDFORD
TO BOSTON              7
TO CHICAGO             5
TO ST.LOUIS            2
TO LEXINGTON           3
FROM YORK
TO BOSTON              2
TO CHICAGO             5
TO ST.LOUIS            4
TO LEXINGTON           5
```

From	To Destination				Supply
	BOSTON	CHICAGO	ST.LOUIS	LEXINGTON	
CLEVELAND	3 0	2 0	7 0	6 0	5000
BEDFORD	7 0	5 0	2 0	3 0	6000
YORK	2 0	5 0	4 0	5 0	2500
	6000	4000	2000	1500	13500

From	To Destination				Supply
	BOSTON	CHICAGO	ST.LOUIS	LEXINGTON	
CLEVELAND	3 5000	2 0	7 0	6 0	0
BEDFORD	7 0	5 0	2 0	3 0	6000
YORK	2 0	5 0	4 0	5 0	2500
	1000	4000	2000	1500	8500

From	To Destination				Supply
	BOSTON	CHICAGO	ST.LOUIS	LEXINGTON	
CLEVELAND	3 5000	2 0	7 0	6 0	0
BEDFORD	7 1000	5 0	2 0	3 0	5000
YORK	2 0	5 0	4 0	5 0	2500
	0	4000	2000	1500	7500

From	To Destination				Supply
	BOSTON	CHICAGO	ST.LOUIS	LEXINGTON	
CLEVELAND	3 5000	2 0	7 0	6 0	0
BEDFORD	7 1000	5 4000	2 0	3 0	1000
YORK	2 0	5 0	4 0	5 0	2500
	0	0	2000	1500	3500

```
..................................................
From              To Destination                 Supply
          BOSTON     CHICAGO    ST.LOUIS  LEXINGTON
CLEVELAND              3          2          7          6
                    5000          0          0          0          0

BEDFORD                7          5          2          3
                    1000       4000       1000          0          0

YORK                   2          5          4          5
                       0          0          0          0       2500

                       0          0       1000       1500       2500
..................................................

..................................................
From              To Destination                 Supply
          BOSTON     CHICAGO    ST.LOUIS  LEXINGTON
CLEVELAND              3          2          7          6
                    5000          0          0          0          0

BEDFORD                7          5          2          3
                    1000       4000       1000          0          0

YORK                   2          5          4          5
                       0          0       1000          0       1500

                       0          0          0       1500       1500
..................................................

                 ROUTE                    UNIT   PER UNIT       TOTAL
FROM                TO                 SHIPPED     COST          COST
CLEVELAND        BOSTON                   5000        3        15000.000
BEDFORD          BOSTON                   1000        7         7000.000
BEDFORD          CHICAGO                  4000        5        20000.000
BEDFORD          ST.LOUIS                 1000        2         2000.000
YORK             ST.LOUIS                 1000        4         4000.000
YORK             LEXINGTON                1500        5         7500.000
                                                     TOTAL      55500
>
```

CHAPTER III

DECISION TREE

3.1 Basic Concepts

The decision tree is one tool that can be used to assist managers in making decisions under uncertainty. For example, consider the situation involving the following conditions.

1. A decision maker may have a finite number of courses of action available.

2. A finite number of possible outcomes for each section may be involved.

3. A loss may be incurred for each action.[5]

The decision maker would naturally like to choose an action that would minimize loss, regardless of the actual states of nature. For example, on any given morning consider the options of taking or not taking a raincoat to work. Thus, there are two courses of action available. Likewise, there are two possible states of nature: it will rain or it will not rain. If he takes a raincoat and it rains, then he will have no loss incurred. However, if he takes a raincoat and it does not rain, he will have a loss of 3 units.

---

[5]Billy E. Gillett, Introduction to Operation Research (New York: McGraw-Hill Book Company, 1976), p. 383.

If he does not take a raincoat and it rains, a loss of 6 units is incurred. Finally, if he does not take a raincoat and it does not rain there is no loss. Table 7 illustrates the courses of action, the states of nature, and the corresponding losses. In this example, no single action would minimize the loss for all states of nature.

TABLE 7

STATUS OF ACTION

ACTIONS	STATE OF NATURE	
	RAIN	NO RAIN
Take raincoat	0	3
Do not take raincoat	6	0

Basically, the amount of uncertainty associated with a problem influences the approach used by the decision maker. For example, if the states of the nature are known with certainty for each action, and if the corresponding losses are known or can be calculated, then the problem is easily solved. The decision-making process in this case is called decision making under certainty. If only the probability of each state of nature for each action is known, then the process is called decision making under risk. On the other hand, if a course of action must be chosen without any

knowledge of the states of nature, then this is a decision making under uncertainty.

Consider the general framework of the decision problem:

suppose $A = a_1, a_2, \ldots a_m$ is the class of all possible action available to the decision maker,

$\theta = \theta_1, \theta_2, \ldots \theta_n$ the class of all states of nature, and

(a_i, θ_j) the loss from using action a_i when the true state of nature is θ_j.

We describe the decision making process through the table 8.

TABLE 8

GENERAL DECISION TABLE

	STATE OF NATURE			
ACTION	θ_1	θ_2	θ_j	θ_n
a_1	$l(a_1,\theta_1)$	$l(a_1,\theta_2)$.....	$l(a_1,\theta_j)$.....	$l(a_1,\theta_n)$
a_2	$l(a_2,\theta_1)$	$l(a_2,\theta_2)$.....	$l(a_2,\theta_j)$.....	$l(a_2,\theta_n)$
a_j	$l(a_j,\theta_1)$	$l(a_j,\theta_2)$.....	$l(a_j,\theta_j)$.....	$l(a_j,\theta_n)$
a_m	$l(a_m,\theta_1)$	$l(a_m,\theta_2)$.....	$l(a_m,\theta j)$.....	$l(a_m,\theta_n)$

The decision maker would like to choose an action that would minimize the loss, regardless of the true status of nature. Usually this is not possible. Hence, the decision maker must resort to methods that will keep the loss at a minimum, based on the amount of available information about the state of nature.[6]

3.2 Decision Tree

Decision problems involving a modest number of decision alternatives and a modest number of states of nature can be analyzed by using a graphical representation of the decision-making process. This is called a decision tree.

A decision tree represents a sequence of decisions or chance happenings. Nodes in the tree represent decisions or chance happenings. Branches from the nodes represent the posssible outcomes. Terminal nodes have assigned values. Branches leading from chance nodes have assigned probabilities. The problem is to select choices to maximize the expected value outcome. Fig. 3 shows the graphical of decision tree.

The decision tree represents a sequence of decisions. The action node has m branches, each leading to a chance node. Each branch represents one of the actions a_j. Each chance node has n branches connections, each leading to a terminal node. Each branch represents the chance happening θ_n. The terminal nodes have values from the decision table.

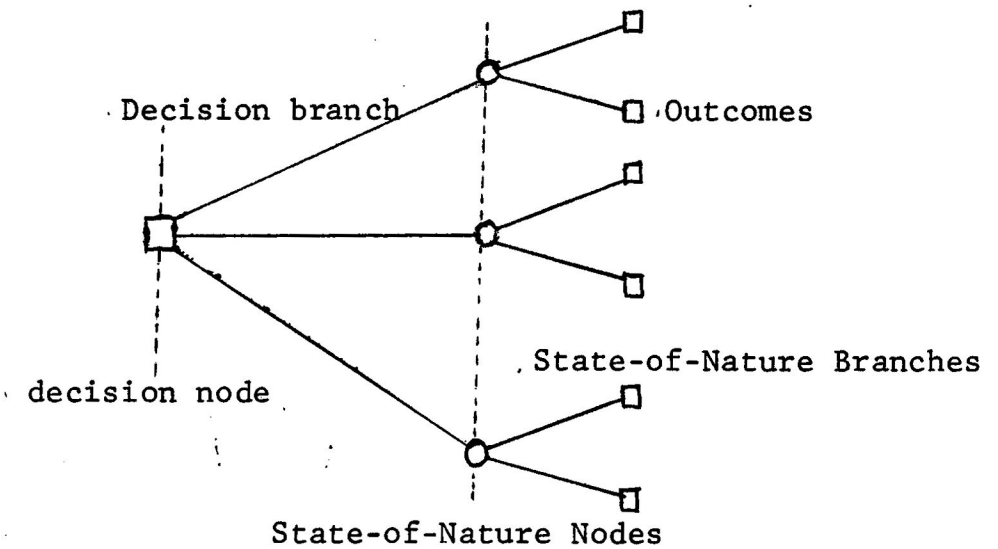
---

[6]Ibid., p. 384.

We can construct a decision tree model of the decision table (page 37) as follows. The root node represents the action to be taken. Each branch from the root represents one of the actions, a_j. These initial branches each lead to a chance node. Each chance node has n branches leading from it representing the chance outcomes. Each branch is labelled with the probability of that outcome. A terminal node is labelled $-l(a_j, \theta_j)$ is if the path to that node is action a_j followed by chance outcome θ_j. Note that the label is $-l(a_j,\theta_j)$ which transforms the minimization problem into a maximizing problem.

FIGURE 3

GRAPHICAL OF DECISION TREE



3.3 Decision Tree Algorithm

Step 1 : Input the branches, the associated branch points and the associated costs.

Step 2 : Start from the open or the terminal nodes of the tree. Calculate the associated gain or loss with each branch.

Step 3 : At each terminal node, put the cost into the associated evaluation matrix. At each state-of-nature node calculate the weighted sum over all paths. At each decision node calculate the optimal cost over all the branches and output the result. Repeat this step until the root of the tree is reached.

Step 4 : Print out the associated gain or loss for the optimal solution.

3.4 Computer Algorithm for Decision Tree

This program is a simple approach to list the branch connections, their expected outcomes and probabilities to generate a tree and its associated branches to allow for the selection of events that will bring about an optional return for a given multiplicity of choices. Terminal events (outcome nodes) are decided by a zero branching point. Chance events are represented by a positive node value generating a decision. All non-terminal events with their associated probabilities and all terminal events with their associated costs are given.

3.4.1 Specified Variables

The specified variables used are listed below :

VARIABLE		DESCRIPTIONS
I,P	=	Number of node in the tree.
2*K	=	Maximum branches per node.
V [I]	=	Value of the I^{th} node.
T [I,j]	=	The storage for decision tree Each value of I represents a node in the tree. I = 0 is the root node.

if T [I,0] = 0 then node I is an outcome and
T [I,1] is its value
if T [I,0] > 0 then I represents a chance node.
T [I,0] is the number of branches.
For each K from 1 to T [I,0] ,
T [I,2*K] is the K^{th} child node
and T [I,2*K-1] is the probability
of that branch.
if T [I,0] < 0 then I represents a decision node
-T [I,0] is the number of branches.
For each K from 1 to T [I,0] ,
T [I,2*K] is the K^{th} child node.

3.4.2 Space Complexity

Theorem : The space complexity of decision tree algorithm is given by O(NK) where N is the number of nodes and K is the maximum number of branches per node.

Proof : The space complexity can be calculated from table 9.

TABLE 9

VARIABLE LIST FOR DECISION TREE METHOD

VARIABLE	TOTAL
0..N OF REAL : V	N+1
0..N, 0..2K OF REAL : T	(2K+1)(N+1)
	2NK+N+2K+3

The total shows that the space complexity of the decision tree is given by O(NK) where N is the number of nodes and K is the maximum of branches per node respectively.

3.4.3 Time Complexity

Theorem : The time complexity of the deicsion tree (ignoring input and output parts) is given by O(P.K) where P is the number of nodes and K is the maximum number of branches per node respectively.

Proof : Various aspects of algorithm for calculation of time complexity are shown on page 46. The calculations and results of time complexity are listed in table 10.

```
PROGRAM DECISION
  BEGIN
   I ← 0 ;
    REPEAT
     READ (K) ; /* number of branches at this
                   node */
     IF K <> 999 THEN
      BEGIN
      T [I,0] ← K ;
      K1 ← ABS (K) ;
  1_2  FOR j : = 1 TO 2*K+1 - SGN (K1) DO
  1_1  READ (T[I,j] ) ; /* Time and possible */
       END ;
     I ← I + 1 ;
    UNTIL (I = 100) OR ( K = 999) ;
    P ← I - 1 ;
    FOR I : = P  DOWNTO Ø DO
     BEGIN
     K ← TRUNC (T [I,1] ) ;
     K2 ← SGN ( K ) ;
     CASE  K2 OF
  1_7  0 :  V [I] ← T [I,1] ; /* move value into
                                 tree */
       1 :  BEGIN
            /* sum up products of probability and
               associated cost */
1_10 1_9 1_8 S ← 0 ;
            j ← 2 ;
       WHILE j < = 2 * K  DO
        BEGIN
        B ← TRUNC (T [I,J] ) ; /* Branch */
  1_4 1_3 S ← S+T [I,j - 1] * V [B] ;
        j ← j+2 ;
        END;
        V [I] ← S ;
      END;
```

```
                                    1 : BEGIN
                                        /* Find optimal cost assoicated
                                           with this branch */
                                        M ← [V  TRUNC (T [I,2]  )]  ;
                                        j ← 2 ;
                                       WHILE j   <= 2 *ABS(B)  DO
                                         BEGIN
                                         V1 ← V [TRUNC (T [I,j] )]  ;
                                         IF   M < V1  THEN /* save value */
                                         BEGIN
l10 l9 l8   l6   l5                      A ← TRUNC (T [I, j -1]  )
                                               /* save action */
                                         END ;
                                        j ← j + 2 ;
                                       END ;
                                         V [I] ← M ;
                                         PRINT A ;
                                    END ;
                                END ;
                            END ;
                        END ;
```

TABLE 10

TIME COMPLEXITY FOR DECISION TREE METHOD

STEP	LEVEL	TYPE OF LOOP	RANGE	ASSIGNMENT AND OPERATION	RESULT	COMMENTS
	l_1	FOR	1..2K	0	0	K = Maximum number of branches per node.
	l_2	REPEAT	UNTILL999	4	4 P	
	l_3	WHILE	2..2K	6	6 K	P = Number of nodes. Maximum is 100.
	l_4	—	—	$3+ l_3$	6K+3	
	l_5	WHILE	2...2K	5	5 K	
	l_6	—	—	—	5K+3	
	l_7	—	—	—	1	
	l_8	CASE	—	$l_4+l_6+l_7$	11K + 7	
	l_9	FOR	P..O	$l_8 + 2$	(11K+9)(P+1)	
	l_{10}	—	—	$l_2+l_3+l_9$	4P + 6 K + (11K + 9)(P+1)	
* TOTAL RESULT					11PK +13 P + 17K + 9	

The total result shows that the time complexity of the decision tree is O(P.K) where P is the number of nodes and K is the maximum branches per node respectively.

```
/*
                Program Documentation
                ---------------------

        Program :       DECISI.PAS
        Author  :       Thaweesak Chamchoi
        Purpose :       Decision Tree
                        Used in selecting from a network of possible
                        decisions, those decisions which will give
                        the maximum return at the greatest probability.
                        it given the probabilities and values as data.

                        Variable List
                        -------------

A               -       Associated value for probability branch
B               -       work variable
FNAME           -       File or Device name
I               -       Index Variable
INF             -       File channel text
J               -       Index Variable
K               -       Index Variable
K1              -       Index variable (Absolute value of K)
K2              -       index variable
KB              -       Test flag for keyboard input device
M               -       Maximum branch for probability loop
P               -       Number of tasks
S               -       Summation of probability
T               -       Matrix containing probability decision data
V               -       Array for storing value of probability branches
V1              -       work variable

                        Procedure List
                        --------------

        SGN             -       Signum function, returns the sign of the
                                input parameter as -1 if negative.0 if
                                zero and 1 is number is positive.

                        INPUT PARAMETERS
                        ----------------

        FOR EACH NODE OF DATA
             IF THE NODE IS A DECISION NODE
               1ST NUMBER IS THE NEGATIVE NUMBER OF BRANCH. I.E. -N
                FOR EACH BRANCH , WE MUST HAVE :
                  (1) SUCCESSOR NODE NUMBER
                  (2) DECISION TASK #

             IF THE NODE IS A STATE-OF-NATURE NODE
               1ST NUMBER IS NUMBER OF BRANCHES,
                FOR EACH BRANCH , WE MUST HAVE :
                  (1) SUCCESSOR NODE #
                  (2) PROBABILITY FOR BRANCH TO HAPPEN
                       NOTE : #2 SHOULD BE BETWEEN 0 AND 1 AND
                              THE SUM OF ALL THE PROBABILITIES SHOULD
                              EQUAL ONE (1).

             IF THE NODE IS AN OUTCOME NODE (TERMINATING NODE)
               1ST NUMBER SHOULD BE ZERO (0)


               2ND NUMBER SHOULD BE EXPECTATION VALUE.
*/
```

```
  PROGRAM DECISION (INPUT,OUTPUT);
VAR
  I,J,K,P,K1,K2,R,A :INTEGER;
  M,V1,S :REAL;
  V :ARRAY[0..100] OF REAL;
  T :ARRAY[0..100,0..10] OF REAL;
  KB:BOOLEAN;
  INF:TEXT;
  FNAME:ARRAY[1..6] OF CHAR;

  FUNCTION SGN(K:INTEGER):INTEGER;
   BEGIN
     IF K<0
      THEN SGN:=-1
      ELSE
      IF K>0
       THEN SGN:=1
       ELSE SGN:=0;
   END;
 BEGIN
   I:=0;
/*       Get Input Medium          */
   WRITE('Enter Input File Name : ');
   READLN(FNAME);
   RESET(INF,FNAME,'.DAT');
/*       Check for Keyboard Input          */
   KB:=(FNAME[1]='K') AND (FNAME[2]='B');
    REPEAT
/* Get data for number of branches */
     IF KB
      THEN
      WRITE('INPUT YOUR BRANCH CONNECTOR ');
     READLN(INF,K);
     IF K<>999
      THEN
       BEGIN
         I[I,0]:=K;
         K1:=ABS(K);
/*       Get Branch Datas - Node, Probability  or Value */
        FOR J:=1 TO 2*K1+1-SGN(K1) DO
         BEGIN
           IF KB
            THEN
            WRITE('ENTER VALUE FOR ASSOC. PATH',J:4,' ');
           READLN(INF,I[I,J]);
         END;
       END;
     I:=I+1;
    UNTIL (I=101) OR (K=999);
   P:=I-1;
```

```
WRITE('Selected Arts ');
/*Starting from the last decisions calculate decision branch values */
   FOR I:= P DOWNTO 0 DO
    BEGIN
      K:=TRUNC(T[I,0]);
      K2:=SGN(K);
      CASE K2 OF
/*      If zero, last node of decision branch
        move value into array
*/
        0:V[I]:=T[I,1];
/*
        If branch node is positive, we have chance data
        so sum up the values of the decision paths coming from
        this node. Note : we must use the probabilities as weights
        for this sum.
*/
        1:
         BEGIN
           S:=0;
           J:=2;
           WHILE J<=2*K DO
            BEGIN
              B:=TRUNC(T[I,J]);
              S:=S+T[1,J-1]*V[B];
              J:=J+2;
            END;
           V[I]:=S;
         END;
        -1:/*
        If branch node is negative, an action must be done.therefore
        find the branch with the maximum value ,and print out as selected
*/

         BEGIN
           M:=V[TRUNC(T[I,2])];
           J:=2;
           WHILE J<=2*ABS(K) DO
            BEGIN
              V1:=V[TRUNC(T[I,J])];
              IF M<=V1
               THEN
                BEGIN
                  M:=V1;
                  A:=TRUNC(T[I,J-1]);
                END;
              J:=J+2;
            END;
           WRITE(A:4);
           V[I]:=M;
         END;
       END;
      /* CASE */
    END;
   /* FOR I */
   WRITELN;
   WRITELN;
/* At end print out total expected gain(loss) */
   WRITELN('EXPECTED GAIN = ',V[0]:8:2);
 END.
```

```
Enter Input File Name : KB:


INPUT YOUR BRANCH CONNECTOR -3
ENTER VALUE FOR ASSOC. PATH   1 1
ENTER VALUE FOR ASSOC. PATH   2 1
ENTER VALUE FOR ASSOC. PATH   3 2
ENTER VALUE FOR ASSOC. PATH   4 2
ENTER VALUE FOR ASSOC. PATH   5 3
ENTER VALUE FOR ASSOC. PATH   6 3
INPUT YOUR BRANCH CONNECTOR 2
ENTER VALUE FOR ASSOC. PATH   1 .5
ENTER VALUE FOR ASSOC. PATH   2 4
ENTER VALUE FOR ASSOC. PATH   3 .5
ENTER VALUE FOR ASSOC. PATH   4 5
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 0
INPUT YOUR BRANCH CONNECTOR 3
ENTER VALUE FOR ASSOC. PATH   1 .333
ENTER VALUE FOR ASSOC. PATH   2 6
ENTER VALUE FOR ASSOC. PATH   3 .167
ENTER VALUE FOR ASSOC. PATH   4 7
ENTER VALUE FOR ASSOC. PATH   5 .5
ENTER VALUE FOR ASSOC. PATH   6 8
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 100
INPUT YOUR BRANCH CONNECTOR -2
ENTER VALUE FOR ASSOC. PATH   1 4
ENTER VALUE FOR ASSOC. PATH   2 9
ENTER VALUE FOR ASSOC. PATH   3 5
ENTER VALUE FOR ASSOC. PATH   4 10
INPUT YOUR BRANCH CONNECTOR 2
ENTER VALUE FOR ASSOC. PATH   1 .5
ENTER VALUE FOR ASSOC. PATH   2 11
ENTER VALUE FOR ASSOC. PATH   3 .5
ENTER VALUE FOR ASSOC. PATH   4 12
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -10
INPUT YOUR BRANCH CONNECTOR 2
ENTER VALUE FOR ASSOC. PATH   1 .75
ENTER VALUE FOR ASSOC. PATH   2 13
ENTER VALUE FOR ASSOC. PATH   3 .25
ENTER VALUE FOR ASSOC. PATH   4 14
INPUT YOUR BRANCH CONNECTOR 3
ENTER VALUE FOR ASSOC. PATH   1 .167
ENTER VALUE FOR ASSOC. PATH   2 15
ENTER VALUE FOR ASSOC. PATH   3 .333
ENTER VALUE FOR ASSOC. PATH   4 16
ENTER VALUE FOR ASSOC. PATH   5 .5
ENTER VALUE FOR ASSOC. PATH   6 17
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -100
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 90
INPUT YOUR BRANCH CONNECTOR -2
ENTER VALUE FOR ASSOC. PATH   1 6
ENTER VALUE FOR ASSOC. PATH   2 18
ENTER VALUE FOR ASSOC. PATH   3 7
ENTER VALUE FOR ASSOC. PATH   4 19
```

```
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 90
INPUT YOUR BRANCH CONNECTOR -2
ENTER VALUE FOR ASSOC. PATH   1 8
ENTER VALUE FOR ASSOC. PATH   2 20
ENTER VALUE FOR ASSOC. PATH   3 9
ENTER VALUE FOR ASSOC. PATH   4 21
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 100
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 0 .
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -100
INPUT YOUR BRANCH CONNECTOR 3
ENTER VALUE FOR ASSOC. PATH   1 .167
ENTER VALUE FOR ASSOC. PATH   2 22
ENTER VALUE FOR ASSOC. PATH   3 .333
ENTER VALUE FOR ASSOC. PATH   4 23
ENTER VALUE FOR ASSOC. PATH   5 .5
ENTER VALUE FOR ASSOC. PATH   6 24
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -110
INPUT YOUR BRANCH CONNECTOR 3
ENTER VALUE FOR ASSOC. PATH   1 .167
ENTER VALUE FOR ASSOC. PATH   2 25
ENTER VALUE FOR ASSOC. PATH   3 .333
ENTER VALUE FOR ASSOC. PATH   4 26
ENTER VALUE FOR ASSOC. PATH   5 .5
ENTER VALUE FOR ASSOC. PATH   6 27
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -110
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 90
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -10
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -110
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 90
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -10
INPUT YOUR BRANCH CONNECTOR 0
ENTER VALUE FOR ASSOC. PATH   1 -110
INPUT YOUR BRANCH CONNECTOR 999


Selected Acts    8   6   4   3

EXPECTED GAIN =    34.44
>
```

BIBLIOGRAPHY

Anderson, R. David; Sweeney, J. Dennis; Williams, A. Thomas. An Introduction to Management Science. St. Paul, Minnesota: West Publishing Co., 1979.

Gillett, E. Billy. Introduction to Operation Research A Computer-Oriented Aleorithmic Approach. New York: McGraw-Hill, Inc., 1976.

Grogono, Peter. Programming in PASCAL. Philippines: Addison-Wesley Publishing, 1980.

Jenson, Kathleen; Wirth, Niklaus. PASCAL User Manual and Report. New York: Springer-Verlag, 1978.

Keheny, C. John; Kurtz, E. Thomas. Basic Programming. New York: John Wiley and Son, Inc., 1976.

Reingold, M. Edward; Nieverbert, Jong; Deo, Nakstneh. Combinatorial Algorithms. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.

Ralston, Anthony; Meek, L. Chester. Encyclopedia of Computer Science. New York: Van Nostrand Reinhold Company, 1976.

Thesen, Akne. Computer Methods in Operations Research. New York: Academic Press, 1978.